ACM321 Java Lab10 - Exceptions

Exceptions

One way in which to write a program is to assume that everything proceeds smoothly and as expected—users input values at the correct time and of the correct format, files are never corrupt, array indices are always valid and so on. Of course this view of the world is very rarely accurate. In reality, unexpected situations arise that could compromise the correct functioning of your program.

Programs should be written that continue to function even if unexpected situations should arise.

So far we have tried to achieve this by carefully constructed if statements that send back error flags, in the form of boolean values, when appropriate. However, in some circumstances, these forms of protection against undesirable situations prove inadequate. In such cases the exception handling
facility of Java must be used.

Pre-defined Exception Classes in Java

Each type of event that could lead to an exception is associated with a pre-defined exception class in Java. When a given event occurs, the Java run-time environment determines which exception has occurred and an object of the given exception class is generated. This process is known as **throwing** an exception.

These exception classes have been named to reflect the nature of the exception. For example, when an array is accessed with an illegal index an object of the **ArrayIndexOutOfBoundsException** class is thrown.

```
public class ArrayBoundException {

  public static void main(String[] args) {
    int[] arr = {0,1,2,3,4,5,6};

      for(int i=0;i<=arr.length;i++) {
        System.out.println(arr[i]);
      }
```

```
    }

}
```

```
<terminated> ArrayBoundException [Java Application] C:\Program Files\Java\jre1.8.0_221\bin\javaw.exe (Apr 24, 2020, 5:03:14 AM)
0
1
2
3
4
5
6
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 7
        at ArrayBoundException.main(ArrayBoundException.java:8)
```

All exception classes inherit from the base class Throwable which is found in the java.lang package. These subclasses of Throwable are found in various packages and are then further categorized depending upon the type of exception.

For example, the exception associated with a given file not being found (**FileNotFoundException**) and the exception associated with an end of file having been reached (EOFException) are both types of input/output exceptions (IOException), which reside in the java.io package.

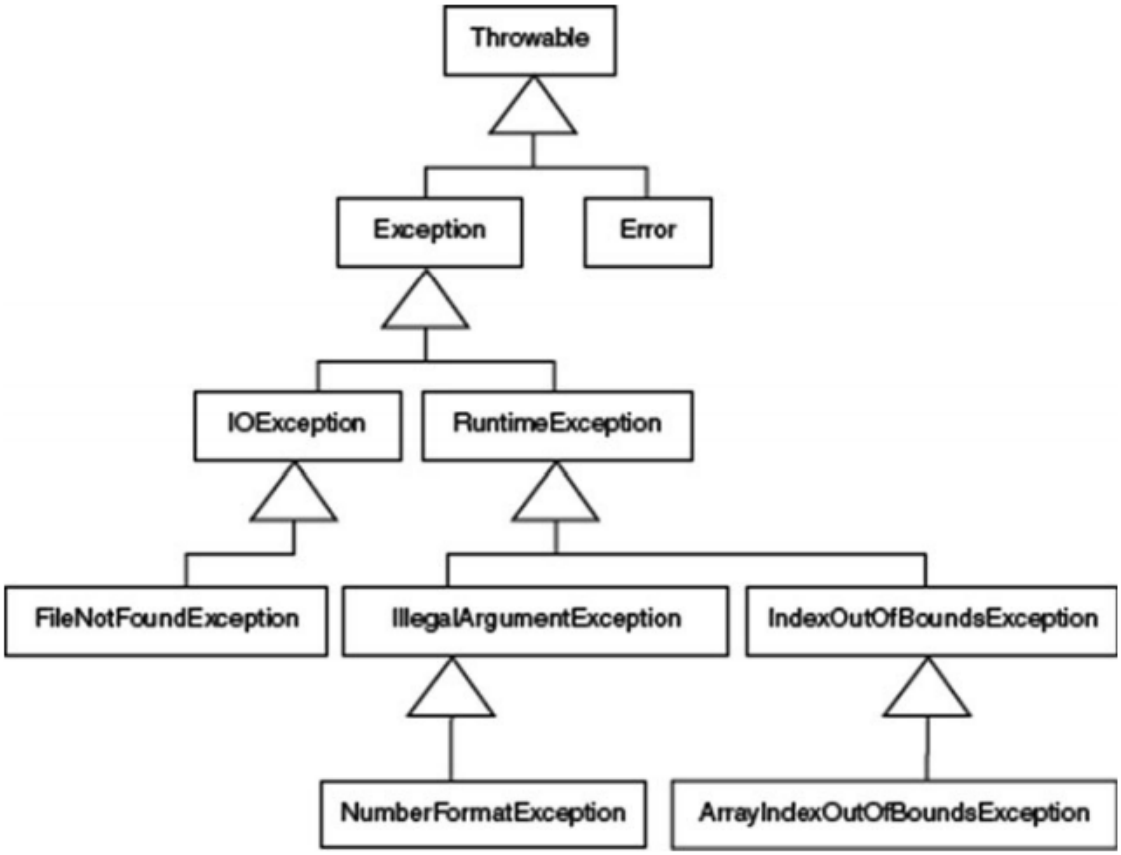

Fig. 14.1 A sample of Java's pre-defined exception class hierarchy

try and catch

The `try` statement allows you to define a block of code to be tested for errors while it is being executed.

The `catch` statement allows you to define a block of code to be executed, if an error occurs in the try block.

```
try {
  //  Block of code to try
}
catch(Exception e) {
  //  Block of code to handle errors
}
```

```
public class TryCatchinClass {

  public static void main(String[] args) {
     try {
          int[] myNumbers = {1, 2, 3};
          System.out.println(myNumbers[10]);
        } catch (Exception e) {
          System.out.println("Something went wrong.");
        } finally {
      System.out.println("The 'try catch' is finished.");
    }
  }
}
```

```
public class TryCatchinClass {
  static void checkAge(int age) {
    if (age < 18) {
      throw new ArithmeticException("Access denied - You must be at least 18 years old.");
    }
    else {
      System.out.println("Access granted - You are old enough!");
    }
  }

  public static void main(String[] args) {
    checkAge(15); //try also for 20
  }

}
```

Working With Files

1. **FileWriter**: FileWriter is the simplest way to write a file in Java. It provides overloaded write method to write int, byte array, and String to the File. You can also write part of the String or byte array using FileWriter. FileWriter writes directly into Files and should be used only when the number of writes is less.
2. **BufferedWriter**: BufferedWriter is almost similar to FileWriter but it uses internal buffer to write data into File. So if the number of write operations is more, the actual IO operations are less and performance is better. You should use BufferedWriter when the number of write operations is more.

```
//WritewithFileWriter.java

import java.io.FileWriter;
```

```
import java.io.PrintWriter;

import java.util.ArrayList;
import java.util.List;

import java.io.IOException;

public class WritewithFileWriter {

  public static void main(String[] args) {
    List<String> carList = new ArrayList<>();
    carList.add("Car A");
    carList.add("Car B");
    carList.add("Car C");

    try {
    FileWriter carFile = new FileWriter("Cars.txt", true);

    PrintWriter carWriter = new PrintWriter(carFile);
    for(int i = 0 ; i < carList.size(); i++)
    {
      carWriter.println(carList.get(i));
    }
    carWriter.close();
    System.out.println("Writed");

    } catch(IOException e)
    {
      System.out.println("There was a problem writing the file");
    }

  }

}
```

```
//ReadwithFileReader.java

import java.io.FileReader;

public class ReadwithFileReader {

  public static void main(String[] args) throws Exception{
        FileReader fr=new FileReader("Cars.txt");
        int i;
        while((i=fr.read())!=-1)
        //It is used to return a character in ASCII form. It returns -1 at the end of file.
        System.out.print((char)i);
        fr.close();
  }
}
```

```
//WritewithBuffer.java

import java.util.ArrayList;
import java.util.List;
```

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

public class WritewithBuffer {

  public static void main(String[] args) {
    List<String> carList = new ArrayList<>();
    carList.add("Car A");
    carList.add("Car B");
    carList.add("Car C");

     try (FileWriter writer = new FileWriter("CarsBuffer.txt");
              BufferedWriter bw = new BufferedWriter(writer)) {
        for(int i = 0 ; i < carList.size(); i++)
        {
          bw.write(carList.get(i) + "\n");
        }
        System.out.println("Writed with buffer");
          } catch (IOException e) {
              System.err.format("IOException:", e);
          }
  }

}
```

```
//ReadwithBuffer.java

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class ReadwithBuffer {

  public static void main(String[] args) {
      try {
         String strCurrentLine;

         BufferedReader objReader = new BufferedReader(new FileReader("CarsBuffer.txt"));

         while ((strCurrentLine = objReader.readLine()) != null) {
          System.out.println(strCurrentLine);
         }

         objReader.close();

        } catch (IOException e) {
          //System.out.print("Error");
          e.printStackTrace();
        }
  }
}
```

Extra Example

```
import java.io.File;  // Import the File class
import java.io.FileNotFoundException;  // Import this class to handle errors
import java.util.Scanner; // Import the Scanner class to read text files

import java.io.FileWriter;   // Import the FileWriter class
import java.io.IOException;  // Import the IOException class to handle errors

import java.io.BufferedReader;
import java.io.BufferedWriter;

public class FileReadException {

  public static void main(String[] args) {
    try {
          File myObj = new File("text.txt");
          Scanner myReader = new Scanner(myObj);
          while (myReader.hasNextLine()) {
            String data = myReader.nextLine();
            System.out.println(data);
          }
          myReader.close();
        } catch (FileNotFoundException e) {
          System.out.println("An error occurred.");
          e.printStackTrace();
        }

    //writing to a file
     try {
        //FileWriter myWriter = new FileWriter("text.txt", true);
          BufferedWriter myWriter = new BufferedWriter(
                        new FileWriter("text.txt", true)  //Set true for append mode
                    );

          myWriter.write("Yeditepe");
          myWriter.newLine(); //only awailable on bufferedwriter
          myWriter.append("University");   //or you can Add new line with \n

          myWriter.close();

          System.out.println("Successfully wrote to the file.");

        } catch (IOException e) {
          System.out.println("An error occurred.");
          e.printStackTrace();
        }
  }

}
```

Java Write to File - 4 Ways to Write File in Java - JournalDev

Java provides several ways to write to file. We can use FileWriter, BufferedWriter, java 7 Files and FileOutputStream to write a file in Java. Let's have a brief look at four options we have for java write to

https://www.journaldev.com/878/java-write-to-file

Java Exceptions (Try...Catch)

When executing Java code, different errors can occur: coding errors made by the programmer, errors due to wrong input, or other unforeseeable things. When an error occurs, Java will normally stop

https://www.w3schools.com/java/java_try_catch.asp